New Jersey's Science & Technology University

THE EDGE IN KNOWLEDGE

CS 280
Programming Language Concepts

Languages – Why and How

New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

How, when and why did programming languages evolve?

- All processors support a very small set of simple instructions
- Instructions are represented as bit patterns that control the operation of the machine
- VERY early on, the notion of a stored program (as compared to wired in or fed in from cards or magnetic tape) developed
- VERY early on, simple mnemonic languages were developed to make programming a little bit easier

Assembler Language

- Machine code has a series of possible operations and operands, represented by unique bit patterns
- Some mnemonic representation of the operations helps make things readable
- Symbolic names for resources like processor registers helps a lot.
- Symbolic names for memory locations helps even more

Assembler example

Count the number of 1 bits in a byte at memory location 0x2000

```
            LXI H 2000H;   ;HL points at location 2000H
            MOV A, M;      ;Loads A with the content of M
            MVI C, 08H;    ;Sets up counter for number of bits
            MVI B, 00H;    ;Sets counter to count number of ones
jump2:      RAL;           ;Rotates accumulator left through carry
            JNC jump1;     ;On no carry jumps to jump1:
            INR B;         ;Increases counter B by one
jump1:      DCR C;         ;Decreases counter C by one
            JNZ jump2;     ;When C is not zero jumps to jump2:
            HLT;           ;Terminates the program
```

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Assembler example

```
; ----------------------------------------------------------------------------------------
; Writes "Hello, World" to the console using only system calls. Runs on 64-bit Linux only.
; To assemble and run:
;
;     nasm -felf64 hello.asm && ld hello.o && ./a.out
; ----------------------------------------------------------------------------------------

          global    _start

          section   .text
_start:   mov       rax, 1                  ; system call for write
          mov       rdi, 1                  ; file handle 1 is stdout
          mov       rsi, message            ; address of string to output
          mov       rdx, 13                 ; number of bytes
          syscall                           ; invoke operating system to do the write
          mov       rax, 60                 ; system call for exit
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Higher Levels of Language

- A higher level of abstraction
- Easier to read and write
- Easier to develop systems for more complex uses

NJIT
New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

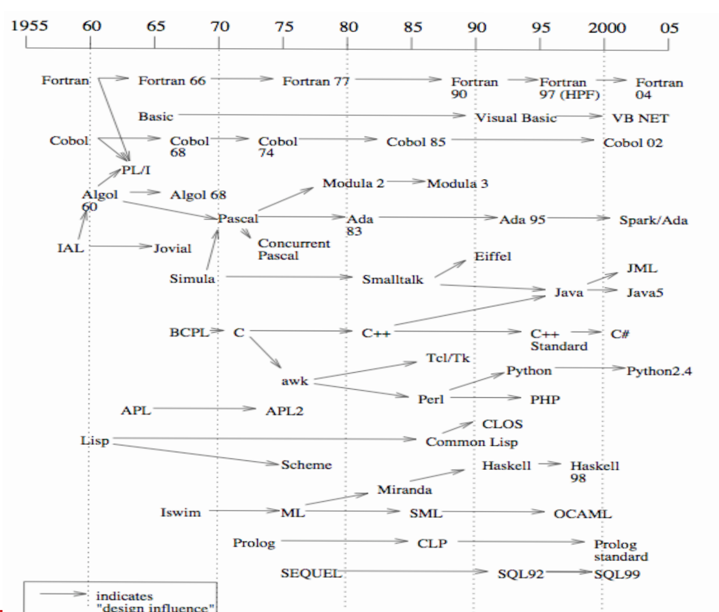
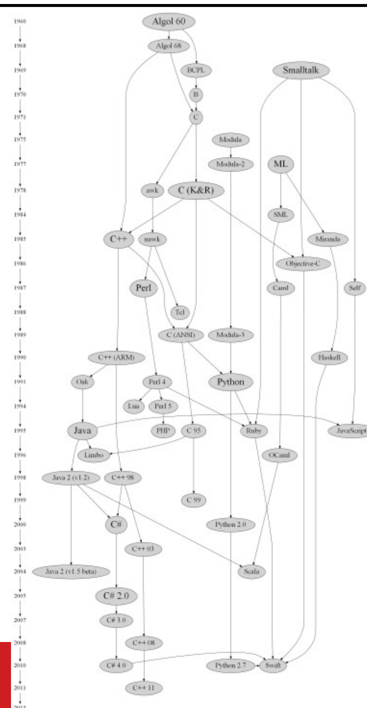


Figure 1.2: A Snapshot of Programming Language History

NJI
New J
ENCES

Programming Language History

C



NJIT New Jersey's Science & Technology University

CES

- Imperative Programming
  - A sequence of statements telling a computer to perform operations
  - Central features are variables, assignment statements, and iteration
  - Includes languages that support object-oriented programming
  - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Declarative Programming
  - Focuses on what to execute, but not how to execute it
  - Rule based languages, logic programming
- Functional Programming
  - Main means of making computations is by applying functions to given parameters
  - Examples: Haskell, Julia, ML

NJIT New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

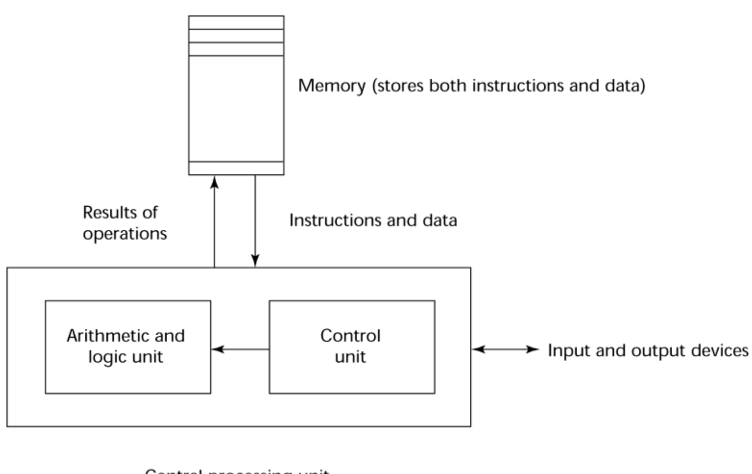
What Is Common

- All Languages will have
  - a syntax – what is and is not valid in the language
  - semantics – what is the meaning of elements of the language
  - names – what are the rules for names for things in the language
  - types – what values can things in the language take on
- All Languages run on some sort of machine

NJIT New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

The von Neumann Architecture



NJIT New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

The von Neumann Architecture

- Fetch-execute-cycle (on a von Neumann architecture computer)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Von Neumann Bottleneck

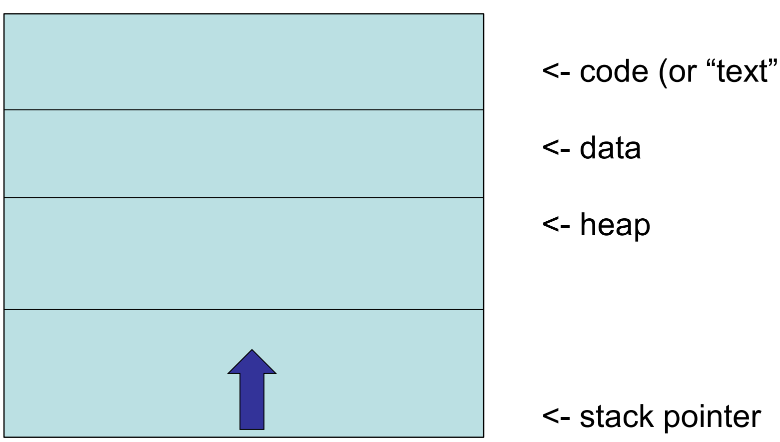
- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed much faster than the speed of the connection; the connection speed thus results in a *bottleneck*
- Known as the *von Neumann bottleneck*; it is the primary limiting factor in the speed of computers

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Memory Usage

- Programs use memory in several ways:
  - the actual executable machine instructions
  - global variables
  - constants (such as strings)
  - memory that is used to support function calls
    - this is usually done in a “stack”, at runtime
  - memory that is dynamically allocated and freed as needed
    - this is usually done in a “heap”, at runtime
    - the “new” operator is one way to dynamically allocate memory

Memory Layout



NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Computer Architecture Influences Languages

- Imperative languages are most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - This is the basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient
  - A language that supports changing memory and conditional branching is said to be "Turing Complete"

Programming Methodologies over Time

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming
  - Data abstraction + inheritance + polymorphism

Language Design Trade-Offs

- Reliability vs. cost of execution
  - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs
- Readability vs. writeability
  Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writeability (flexibility) vs. reliability
  - Example: C++ pointers are powerful and very flexible but can be unreliable

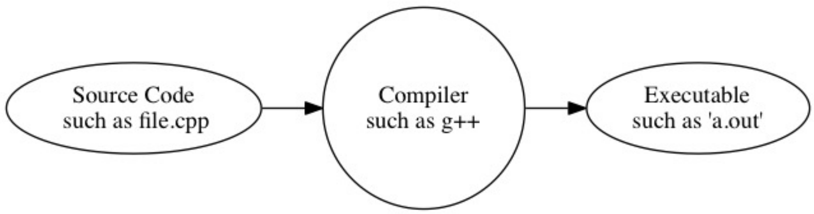
NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Implementation Methods

- Compilation
  - Programs are translated into machine language; includes JIT systems
  - Use: Large commercial applications
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
  - Use: Small programs or when efficiency is not an issue
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters
  - Use: Small and medium systems when efficiency is not the first concern

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Compiling



NJIT New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

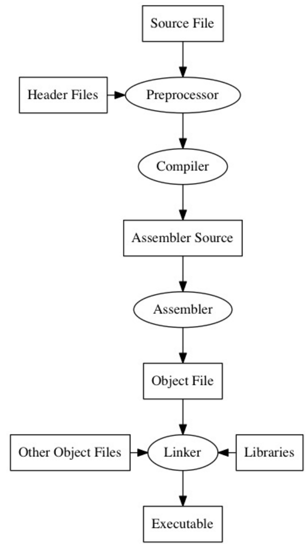
Compilation

- Translate high-level program (source language) into machine code (machine language)
- This is specific to the machine you are compiling for
- Compiling on one machine to run on another is called “cross-compilation”
- Slow translation, fast execution

NJIT New Jersey's Science & Technology University

COLLEGE OF COMPUTING SCIENCES

Steps In Compile



Preprocessors

- Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
- A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
- A well-known example: C preprocessor
  - expands `#include`, `#define`, and similar macros

Object Files

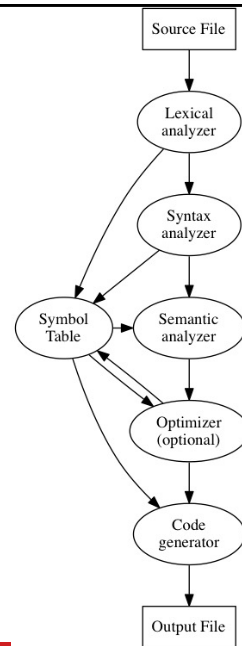
- Each Object File contains:
  - executable code
  - global variables
  - constants
  - "linkage information"
    - what code and data do I provide that others may use
    - what code and data do I require that I need others to provide for me
- A library is just a collection of object files (sometimes an "archive" or a ".a" or a "library" or a ".lib")

Linking

- The "linker" connects all the object files together to make a single executable
- Every required connection muse be resolved or the link fails
- Some environments use a "shared library" scheme
  - On windows this is a .dll, a Dynamically Linked Library
  - On linux it's sometimes a .so, a Shared Object
- With shared libraries, some of the linking happens at run time

Inside a Compiler

- lexical analyzer converts characters in the source program into lexical units
- syntax analyzer transforms lexical units into *parse trees* which represent the syntactic structure of program
- Semantics analyzer enforces the semantic rules of the language
- Optimizer improves the code
- Code generator creates the output (assembler or byte code)



NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately be displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Now rare for traditional high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript, PHP)

NJIT New Jersey's Science & Technology University COLLEGE OF COMPUTING SCIENCES

Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, *byte code*, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called *Java Virtual Machine*)

Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system
- In essence, JIT systems are delayed compilers

NJIT New Jersey's Science & Technology University
COLLEGE OF COMPUTING SCIENCES

Programming Environments

- The compiler/interpreter comes with standard libraries and a collection of other tools used in software development. This is sometimes called the "toolchain"
- Example tools:
  - Linkers
  - Archivers (to make libraries)
  - Symbol readers
  - Debuggers
- Debuggers allow for breakpointing and single stepping through your program, examining the value of variables and monitoring the flow of the program